МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра многопроцессорных систем и сетей

АГРЕГАТОР ЗАПРОСОВ С ЕСТЕСТВЕННОГО ЯЗЫКА: КЛИЕНТСКАЯ ЧАСТЬ

Курсовой проект

Аладко Анастасии Дмитриевны

обучающейся 3 курса

специальности «Прикладная информатика»

Научный руководитель:

старший преподаватель

В.Ю. Сакович

Минск, 2024

РЕФЕРАТ

Курсовой проект 25 с., 3 рис., 4 источн.

НЕЙРОСЕТЬ, ОТОБРАЖЕНИЕ ДАННЫХ, КЛИЕНТСКАЯ ЧАСТЬ, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС, ВЕБ-ПРИЛОЖЕНИЕ, ИНТЕГРАЦИЯ ДАННЫХ, API, SCSS, PARCEL, ДИНАМИЧЕСКОЕ ВЗАИМОДЕЙСТВИЕ

**Объектом исследования является** веб-приложение, предназначенное для объединения результатов работы различных нейросетей, обрабатывающих текстовые, графические и файловые запросы.

**Цель работы – р**азработка интуитивно понятной клиентской части, которая обеспечивает сбор, обработку и отображение данных, полученных от нескольких нейросетей, в формате диалога.

Для реализации проекта использовались технологии веб-разработки, включая HTML, SCSS для стилизации интерфейса и JavaScript для динамической обработки данных. Инструмент Parcel обеспечивал сборку клиентской части, а серверная часть на Python реализовывала взаимодействие с API нейросетей.

**В результате работы** создана функциональная клиентская часть агрегатора запросов, которая объединяет и отображает результаты работы различных нейросетей в одном интерфейсе с использованием современных технологий веб-разработки. Реализован диалоговый интерфейс, обеспечивающий удобное и интерактивное взаимодействие пользователя с системой.

**Область применения результатов** – разработанная система может использоваться для построения платформ на базе искусственного интеллекта в таких сферах, как образование, бизнес, медицина, а также в повседневной жизни для упрощения работы с данными и принятия решений.

ОГЛАВЛЕНИЕ

|  |  |
| --- | --- |
| Термины и определения | 3 |
| Введение | 5 |
| Глава 1 Создание дизайна | 6 |
| * 1. Концепции      1. Общие сведения      2. Дизайн-концепция | 6  6  6 |
| 1.2 Необходимые элементы  1.3 Работа с Figma  1.3.1 Актуальность Figma | 6  7  5 |
| Глава 2 Создание сайта  2.1  2.2  2.3 | 10  45  45 |
| Заключение | 11 |
| Список использованных источников | 12 |
| Приложение А Наименование | 13 |

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

figma — онлайн-сервис для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени

parcel — это инструмент для сборки веб-приложений, который автоматизирует процесс обработки исходного кода, включающего JavaScript, CSS, HTML, изображения и другие ресурсы

Node.js — программная платформа, основанная на движке V8, превращающая JavaScript из узкоспециализированного языка в язык общего назначения

API — это интерфейс программирования приложений, набор правил и механизмов, которые позволяют одному программному обеспечению взаимодействовать с другим

ВВЕДЕНИЕ

Современные технологии искусственного интеллекта и обработки естественного языка стремительно развиваются. Сегодня нейросети могут не только понимать текстовые запросы, но и обрабатывать изображения, файлы и другие форматы данных. Такие системы становятся важной частью различных сфер жизни — от образования и медицины до бизнеса и повседневного общения. Однако важной задачей остаётся создание удобных инструментов, которые могли бы объединять ответы из разных источников и представлять их в интуитивно понятном диалоговом формате.

На данный момент существует множество нейросетей, каждая из которых специализируется на решении определённых задач. Например, одни модели генерируют текстовые ответы, другие — анализируют изображения, а третьи — обрабатывают сложные файлы. Тем не менее, проблема состоит в том, что их работа разрознена, и пользователю приходится обращаться к разным инструментам для получения комплексной информации. Это усложняет взаимодействие и замедляет процесс решения задач.

Исходными данными для выполнения работы являются результаты, предоставляемые различными нейросетями. Новизна проекта заключается в разработке клиентской части системы-агрегатора, которая объединяет данные из текстовых, графических и файловых запросов, а также отображает их в формате диалога, схожего с тем, как это реализовано на современных чат-платформах. Это создаёт единое пространство для взаимодействия пользователя с информацией, поступающей из разных источников.

Таким образом, актуальность работы заключается в необходимости упрощения взаимодействия пользователей с многозадачными системами на базе нейросетей. Разработка такого агрегатора позволит значительно повысить удобство и эффективность использования технологий искусственного интеллекта, предлагая пользователю интуитивно понятный интерфейс для работы с разными форматами данных.

ГЛАВА 1 СОЗДАНИЕ ДИЗАЙНА

1.1 Концепции

1.1.1 Общие сведения

Создание сайта начинается с разработки его концепции. Этот этап включает определение целей сайта, его уникальных черт и преимуществ по сравнению с конкурентами.

«Веб-дизайн включает оформление контента и использование графических элементов на странице. Ранее под дизайном подразумевалось исключительно визуальное оформление, но сегодня акцент сместился на удобство пользователя. Для этого важно анализировать информацию и структурировать её таким образом, чтобы она была понятной и доступной.

Главная задача веб-дизайна — сделать взаимодействие пользователя с сайтом лёгким и приятным. Это повышает удобство использования и положительно сказывается на поведении посетителей» [5].

Когда клиент заходит на сайт, он должен быстро находить необходимую информацию. На первый взгляд задача может показаться простой: расположить контент и основные элементы на странице. Однако успешное выполнение зависит от типа сайта, объёма информации и целей проекта.

Не менее важным аспектом является адаптивный дизайн, который обеспечивает корректное отображение сайта на различных устройствах. Пользователь должен иметь возможность просматривать страницы с компьютера, планшета или телефона, независимо от разрешения экрана.

1.1.2 Дизайн-концепция

Основными функциями моего проекта планируются:

– поддержка текстовых запросов;

– интерфейс для визуализации картинок или изображений, полученных нейросетью;

– история запросов пользователя;

– возможность регистрации и сохранение пользователя.

1.2 Необходимые элементы

Для успешной реализации проекта необходимо предусмотреть следующие элементы:

– **Удобный интерфейс для взаимодействия с пользователем**:

1) поле для ввода запросов и кнопка отправки должны быть простыми и интуитивно понятными;

2) основная область экрана будет использоваться для отображения диалогового формата, где пользователи смогут просматривать историю запросов и их ответы.

– **Бургер-меню**:

1) меню, расположенное в левой части экрана, займёт около 20% пространства и будет содержать историю запросов;

2) возможность скрыть или открыть меню для оптимизации рабочего пространства.

– **Обработка данных из нескольких источников**:

1) система должна интегрироваться с разными нейросетями, чтобы обрабатывать текстовые запросы, анализировать изображения и работать с файлам;

2) все ответы должны быть объединены в единый поток диалога.

– **Адаптивный дизайн**:

1) интерфейс должен корректно отображаться на любых устройствах: компьютерах, планшетах и смартфонах;

2) оптимизация под различные разрешения экрана для обеспечения удобства использования.

– **История запросов**:

хранение всех предыдущих запросов и ответов в формате, доступном для быстрого просмотра и навигации.

Реализация этих элементов позволит создать мощный инструмент для работы с информацией из различных источников, предоставляя пользователям гибкий и удобный интерфейс для взаимодействия.

1.3 Работа в Figma

1.3.1 Актуальность Figma

«Figma — это современный облачный инструмент, предназначенный для совместной работы над дизайном и проектированием интерфейсов» [1]. Благодаря своему удобному и интуитивно понятному интерфейсу, а также мощному функционалу, Figma завоевала популярность среди дизайнеров, разработчиков и менеджеров по всему миру. «Она предоставляет возможности для создания интерфейсов, прототипов, анимаций и тестирования, что делает её незаменимым инструментом в разработке современных цифровых продуктов» [1].

Одной из ключевых особенностей Figma является возможность совместной работы в реальном времени. Пользователи могут одновременно редактировать один и тот же проект, отслеживать изменения, оставлять комментарии и мгновенно получать обратную связь. Такая функциональность значительно ускоряет процессы согласования и улучшения дизайна, особенно в командах с удалённой работой. Кроме того, Figma обладает кроссплатформенностью — доступ к проектам можно получить через любой браузер без необходимости установки дополнительного программного обеспечения. Также доступны приложения для Windows, macOS и мобильных устройств, что делает работу ещё более гибкой.

«Figma широко используется в различных сферах. В веб-дизайне она применяется для создания макетов сайтов, интерфейсов и приложений. При разработке программного обеспечения дизайнеры создают UX/UI-проекты, которые затем передаются разработчикам для реализации. Маркетологи используют Figma для создания баннеров, рекламных креативов и других визуальных материалов. Также Figma активно применяется в образовании, где студенты и преподаватели изучают основы дизайна и работают над совместными проектами.

Актуальность Figma обусловлена несколькими важными факторами. Во-первых, она ускоряет процесс разработки. Во-вторых, её универсальность делает инструмент доступным как для новичков, так и для профессионалов. Figma снижает затраты, так как не требует мощного оборудования или дорогостоящего софта, благодаря своей облачной архитектуре. Figma активно поддерживается сообществом, предоставляющим огромное количество обучающих материалов, шаблонов и плагинов, что делает её доступной даже для начинающих пользователей» [6].

Сегодня, в условиях глобальной цифровизации и распространения удалённой работы, Figma стала одним из самых востребованных инструментов для дизайна. Её возможности объединять команды, упрощать процессы проектирования и предоставлять мощные инструменты для создания качественных продуктов делают её незаменимым помощником в мире цифрового дизайна.

1.3.2 Дизайн в Figma

Я использовала Figma для разработки пользовательского интерфейса агрегатора запросов, который объединяет результаты работы нескольких нейросетей. Этот макет служит основой клиентской части проекта и обеспечивает интуитивное взаимодействие пользователя с системой.

Для визуального оформления я выбрала минималистичный стиль. Фон серого цвета помогает сосредоточиться на функциональных элементах интерфейса.

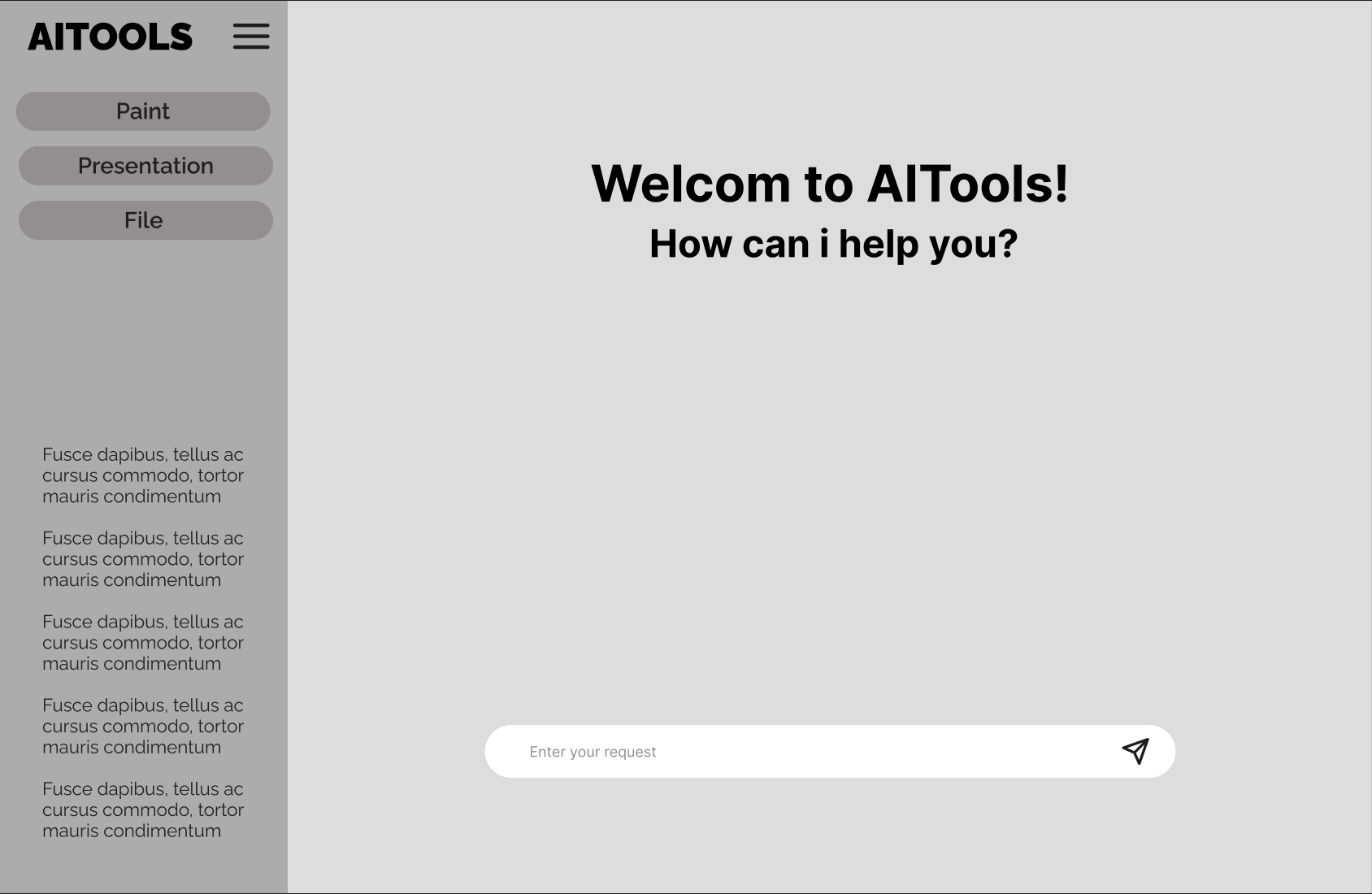
На главной странице макета я предусмотрела две ключевые области. Во-первых, это боковое меню, которое расположено слева и занимает около 20% ширины рабочей области, как видно на рисунке 1.1. Это меню выполняет функцию навигации: в нём хранятся кнопки, такие как «Paint», «Presentation» и «File», а также история последних запросов. Я добавила возможность скрывать и отображать это меню, чтобы пользователь мог эффективно управлять рабочим пространством.

Рисунок 1.1 – Дизайн основного окна

Основная область страницы занимает 80% ширины экрана, что можно заметить на рисунке 1.1. Я сделала её максимально удобной для просмотра информации и взаимодействия с системой. Здесь размещено приветственное сообщение «Welcome to AITools! How can I help you?», которое задаёт дружелюбный тон всему интерфейсу. В дальнейшем данное приветствие будет скрываться и показываться диалоговое окно с запросами, как изображено на рисунке 1.2. В нижней части я разместила поле для ввода текста и кнопку отправки запроса, стилизованную в виде стрелки. Они выделяются на фоне благодаря контрастным цветам, что улучшает восприятие интерфейса. Это упрощает процесс взаимодействия, делая его понятным даже для новых пользователей.

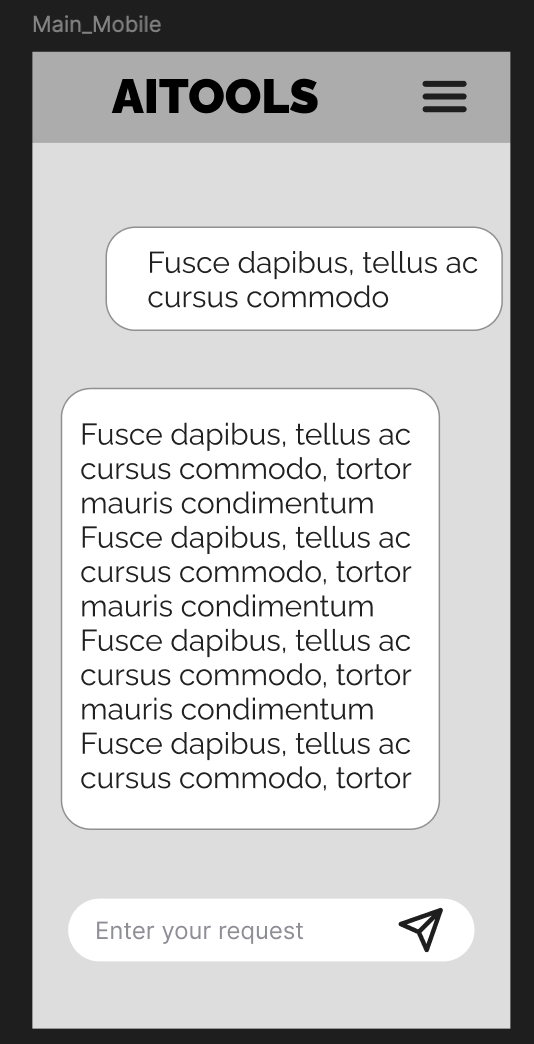
Рисунок 1.2 – Дизайн окна с диалоговым окном

Я также позаботилась о добавлении интерактивных элементов. Например, боковое меню позволяет пользователю переключаться между категориями и просматривать историю запросов. Это особенно важно для тех, кто работает с разными форматами данных, будь то текст, изображения или файлы.

Этот дизайн решает сразу несколько задач. Во-первых, он упрощает взаимодействие пользователя с системой, объединяя все инструменты в одном месте. Во-вторых, он обеспечивает доступ к истории запросов, что делает работу более продуктивной. Наконец, он создаёт адаптивный интерфейс, который корректно отображается на устройствах с разными экранами, будь то настольный компьютер, планшет или смартфон. Дизайн приложения на смартфоне можно увидеть на рисунке 1.3.

Но и это еще не все. В Figma можно посмотреть любой размер, радиус фигуры, цвет, шрифт, размер шрифта. В общем, все стили для сайта, что облегчает дальнейшее написание кода.

Рисунок 1.3 – Дизайн окна с диалоговым окном

Таким образом, разработанный в Figma макет задал структуру и стиль для всего проекта. Он стал не только визуальной основой, но и инструментом для создания функционального и удобного интерфейса, который соответствует задачам агрегатора запросов. С помощью Figma я смогла быстро создать прототип интерфейса, протестировать его и внести необходимые изменения.

ГЛАВА 2 СОЗДАНИЕ САЙТА

2.1 Основные задачи

Основной задачей клиентской части моей работы является создание интерфейса, который будет отображаться сервером на Python и обеспечивать пользователю удобный доступ к функционалу системы. Это предполагает, что все ключевые элементы страницы должны быть тщательно продуманы, а процесс их интеграции с серверной частью — оптимизирован для корректной работы в условиях выбранной архитектуры.

В данном проекте сервер функционирует исключительно с использованием статических файлов, что накладывает определенные требования. Основной задачей становится предварительная сборка всех необходимых файлов (HTML, CSS, JS и других ресурсов) перед запуском сервера. Это гарантирует, что сервер сможет корректно предоставлять пользователю готовые статические файлы без необходимости динамической обработки или трансформации данных на стороне сервера.

Шаблон основной страницы должен включать минимальный набор элементов, необходимых для диалогового интерфейса: **область для отображения сообщений пользователя и ответов системы, форму ввода для запросов, обработка мультимедийных данных.**

Критически важным требованием является оптимизация структуры проекта для сборки. Все файлы должны быть организованы таким образом, чтобы их можно было собрать в единый комплект с минимальными усилиями. Для этого необходимо:

– использовать инструмент сборки, который автоматически компилирует SCSS в CSS и объединяет JavaScript-модули;

– обеспечить четкое разделение кода на модули, что позволит упростить поддержку и добавление нового функционала;

– предусмотреть поддержку современных веб-стандартов, чтобы страница могла корректно работать в большинстве современных браузеров.

Наконец, шаблон страницы должен быть удобным для дальнейшей интеграции с серверной частью. Это означает, что структура HTML должна предусматривать возможность динамического изменения отдельных элементов (например, добавление новых сообщений в диалоге или обновление области мультимедиа).

Таким образом, основные задачи клиентской части включают создание функционального шаблона страницы, который включает ключевые элементы для взаимодействия с пользователем и обеспечение предварительной сборки всех файлов для статической отдачи сервером.

2.2 Выбор среды разработки

Для среды разработки я выбрала Node.js. Этот выбор обусловлен рядом причин, связанных с функциональностью, производительностью и удобством, которые Node.js предлагает мне, как разработчику. «Это популярная серверная среда, основанная на движке v8, которая позволяет выполнять JavaScript-код вне браузера» [4].

Одним из ключевых факторов, определивших выбор Node.js, является ее превосходная поддержка экосистемы JavaScript. В рамках этого проекта в моей части активно используется SCSS для стилизации и Parcel как фреймворк для сборки. Оба этих инструмента беспрепятственно работают с Node.js, так как она предоставляет платформу для установки и использования различных пакетов через npm (Node Package Manager). Благодаря npm, можно легко подключать и обновлять нужные зависимости, что делает процесс разработки быстрым и удобным.

Еще одной причиной выбора Node.js является ее кроссплатформенность. Среда отлично работает на различных операционных системах, включая Windows, macOS и Linux. Это упрощает командную разработку и тестирование проекта в разных средах, что было полезно.

Кроме того, «Node.js предоставляет возможность использовать современные возможности JavaScript, такие как ES-модули и асинхронные функции» [4]. Это позволяет писать чистый и эффективный код, что важно для разработки сложного интерфейса, где обрабатываются диалоги, изображения и файлы. Интеграция с Parcel выполняется нативно, обеспечивая минимальные усилия для настройки. Node.js также поддерживает запуск локального сервера, который необходим для тестирования и отладки клиентской части до ее интеграции с сервером Python.

В рамках проекта Node.js играет роль не только среды разработки, но и инструмента для упрощения управления зависимостями, поддержки модульности и ускорения рабочих процессов. Таким образом, выбор Node.js как среды разработки полностью соответствует задачам проекта, обеспечивая оптимальное сочетание гибкости, производительности и удобства.

2.3 Выбор фреймворка

В моей работе важным аспектом является работа с различными типами данных (текст, изображения, файлы) и отображение диалогового интерфейса, поэтому я выбрала фреймворк Parcel. «Parcel — это современный инструмент для сборки веб-приложений, который сочетает в себе простоту использования, мощные возможности и высокую производительность» [2].

Parcel идеально подходит для проекта, так как он обеспечивает беспрепятственное взаимодействие с современными технологиями фронтенда, такими как SCSS, и предлагает интуитивно понятный процесс настройки. Одним из ключевых преимуществ Parcel является его нативная поддержка модулей JavaScript и множества других форматов. Это упрощает работу с различными типами данных, включая графику, стили и текстовые ресурсы. В моем проекте, где предполагается интеграция с несколькими нейросетями и обработка большого количества пользовательских данных, такая универсальность крайне важна.

Еще одним важным аспектом является простота настройки. Parcel позволяет начать работу без необходимости сложной конфигурации. Это особенно полезно для проекта, где акцент сделан на функциональности и взаимодействии с сервером на Python, а не на детальной настройке клиентской инфраструктуры. Parcel автоматически определяет зависимости, компилирует SCSS в CSS, а также поддерживает современные функции JavaScript, что позволяет использовать последние возможности языка.

Важным аргументом в пользу использования Parcel является его «автоматическое обновление модулей в браузере во время выполнения без необходимости обновления всей страницы(hot module replacement)» [2]. Это позволяет в реальном времени видеть изменения в коде, что ускоряет разработку интерфейса и упрощает тестирование функционала, особенно в рамках динамического диалога, имитирующего взаимодействие с пользователем.

Интеграция Parcel с серверной частью на Python также не вызывает сложностей. Parcel способен генерировать статические файлы, которые легко передаются серверу. Это позволяет сосредоточиться на архитектуре и логике приложения, не отвлекаясь на проблемы совместимости между клиентом и сервером.

Таким образом, выбор Parcel для разработки клиентской части обусловлен его простотой, высокой производительностью, поддержкой современных стандартов веб-разработки и удобством интеграции. Эти преимущества делают Parcel идеальным инструментом для реализации сложного, многозадачного проекта с использованием SCSS и Python-сервера.

2.4 Структура проекта

Если посмотреть на рисунок 2.1, можно заметить, что структура проекта

организована с разделением клиентской и серверной частей, что обеспечивает удобство разработки, тестирования и последующего масштабирования.

Ключевые ресурсы клиентской части хранятся в папке static. Это основной раздел, содержащий все файлы, необходимые для создания и отображения интерфейса. Папка sass играет важную роль в управлении стилями, предоставляя модульное разделение SCSS-файлов, каждый из которых отвечает за стилизацию отдельных элементов интерфейса, таких как заголовок и меню с диалогом. Эти файлы компилируются в финальный CSS-файл, который хранится в папке css и подключается к HTML-шаблонам. Логика клиентского взаимодействия реализована в файле index.js, который также располагается в static/js и отвечает за динамическое поведение страницы, например, обработку пользовательских запросов и обновление интерфейса.

В папке templates находится файл index.html, представляющий собой шаблон основной страницы приложения. Он связывает статические ресурсы с серверной частью, обеспечивая корректное отображение интерфейса. Этот шаблон структурирован таким образом, чтобы его было легко дополнять и масштабировать в будущем.

Серверная часть представлена файлами в корне проекта и папке python. Главный файл сервера — app.py, который отвечает за запуск приложения, обработку HTTP-запросов и передачу HTML-шаблонов вместе со статическими файлами пользователю.

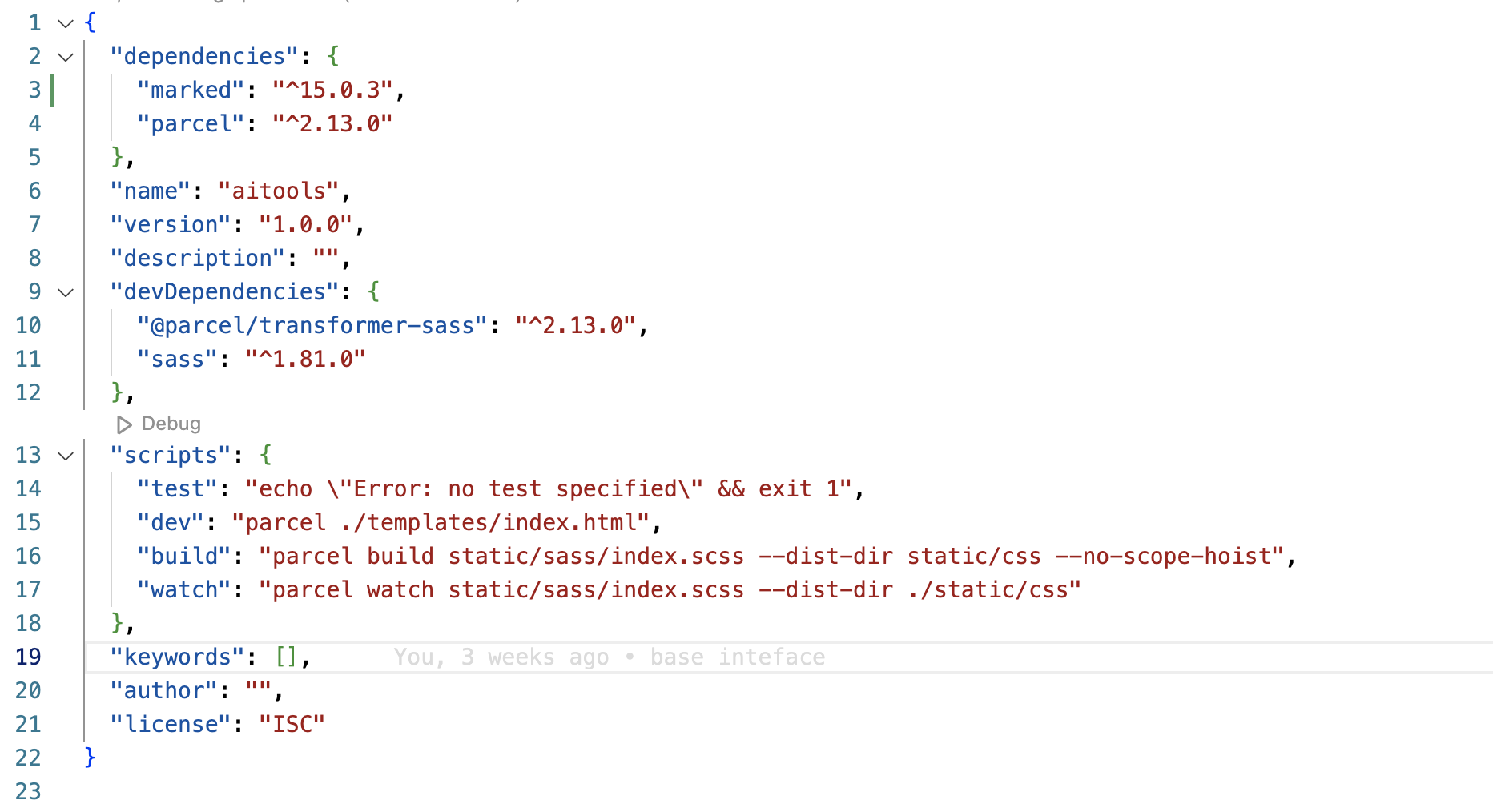
Для работы проекта предусмотрены и вспомогательные папки. Например, node\_modules содержит клиентские зависимости, установленные через npm, которые необходимы для сборки интерфейса. Папка .parcel-cache отвечает за кэширование сборочного процесса Parcel, а dist содержит итоговую сборку файлов клиентской части, готовую к использованию сервером.

Рисунок 2.1 – Структура проекта

2.5 Реализация

2.5.1 Создание проекта Node.js

Реализация проекта началась с установки Node.js, что предоставило доступ к инструменту npm (Node Package Manager). Этот менеджер используется для управления зависимостями и настройкой проекта. После установки была выполнена инициализация проекта с помощью команды npm init. Эта команда создала файл package.json, который является центральным элементом проекта. В этом файле содержится информация о проекте, включая его название, версию, зависимости, а также полезные скрипты для автоматизации задач.

В ходе инициализации через командную строку были заданы базовые параметры проекта, такие как название ("AITools"), версия и точка входа (основной файл проекта). Завершив настройку, в корне проекта появился файл package.json, содержащий структуру, которая облегчает установку и управление зависимостями.

Далее была создана структура директорий и файлов, которая соответствовала потребностям проекта. Основные папки и файлы включали:

– static — для хранения клиентских ресурсов, таких как CSS, JS,

– templates — для хранения HTML-шаблонов, используемых сервером на Python,

– dist — папка для финальной сборки, в которую Parcel будет помещать скомпилированные и оптимизированные файлы,

– python — для серверной логики и обработки запросов, поступающих от клиентской части,

– app.py — основной файл серверного приложения на Python, который отвечает за работу с шаблонами и API.

Также были добавлены системные файлы, такие как .gitignore, чтобы исключить из системы контроля версий временные файлы.

2.5.2 Подключение и установка зависимостей

Одним из ключевых инструментов, выбранных для проекта, стал Parcel. Этот бандлер отличается высокой производительностью и простотой настройки, что делает его идеальным для проектов, требующих гибкой сборки клиентских файлов. Установка Parcel была выполнена с помощью npm, командой npm install parcel --save-dev.

Флаг --save-dev указывает на то, что Parcel будет использоваться только в процессе разработки, а не в финальном продукте. После установки Parcel был добавлен в раздел devDependencies файла package.json. Это позволяет точно отслеживать, какие инструменты необходимы для разработки проекта, и легко восстанавливать окружение на других машинах.

Для работы с CSS был выбран SCSS, более мощная версия CSS, предоставляющая «возможности использования переменных, вложенности, миксинов и других полезных функций» [3]. SCSS позволяет организовать стили проекта модульно и эффективно. Parcel поддерживает SCSS и автоматически компилирует их в итоговый CSS, но для этого требуется минимальная настройка. В проекте были установлены необходимые пакеты, такие как sass и parcel, через npm. После установки SCSS-файлы можно подключать напрямую в проект. Parcel автоматически обрабатывает их при сборке. Для создания финальной версии проекта используется команда npm run build, которая запускает процесс компиляции и оптимизации всех файлов. Таким образом, дополнительных сложных настроек не потребовалось.

2.5.3 Создание и организация HTML

HTML является основой моей части проекта, так как отвечает за структуру и содержание веб-страницы. В рамках реализации проекта был создан файл index.html, представляющий собой центральный шаблон для интерфейса.

Для начала рассмотрим его структуру. На данном фрагменте одним из важным моментов является ссылка на стили. Я писала на scss, но, чтобы запустить проект на python, файлы должны быть статическими, поэтому для подключения CSS и JavaScript используется шаблон Jinja, представленный функцией url\_for в конструкции на 8 и 67 строке. Эта функция динамически формирует URL для статических файлов, обеспечивая их правильное расположение даже при изменении структуры проекта. Это особенно полезно для серверной части, которая загружает и обслуживает данные файлы.

На данном фрагменте кода мы видим секцию для основного контента. Ее разделяем на две логические части: боковое меню и сообщения. В каждой части мы создаем блок с классом «container» для задания общих стилей и автоматического изменения в ходе каких-то перемен.

Далее в боковом меню прописываем название нашего сайта, создаем блок с кнопками одинакового класса для упрощения стилизации. И ниже создаем блок для будущей истории запросов.

В диалоговой части все не так просто. В контейнере у нас два блока и форма. Блок с приветствием в дальнейшем с помощью js будет прятаться для пользователя при начале диалога. В блоке с id = «dialogWrap» на данный момент ничего нет, все будет сгенерировано через js. И форма, в которой у нас лежит input с соответствующим label и кнопка button с тегом i внутри для отображения нашей стрелки отправки с помощью font-awesome.

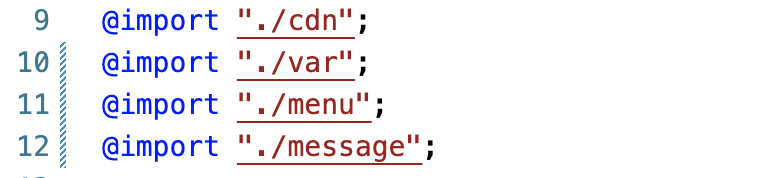
И далее перед закрытием тега body мы указываем наши скрипты. Первый – для оформления приходящих ответов на запрос через markdown, вторая – статический файл index.js и последний – файл для форматирования приходящей строки.

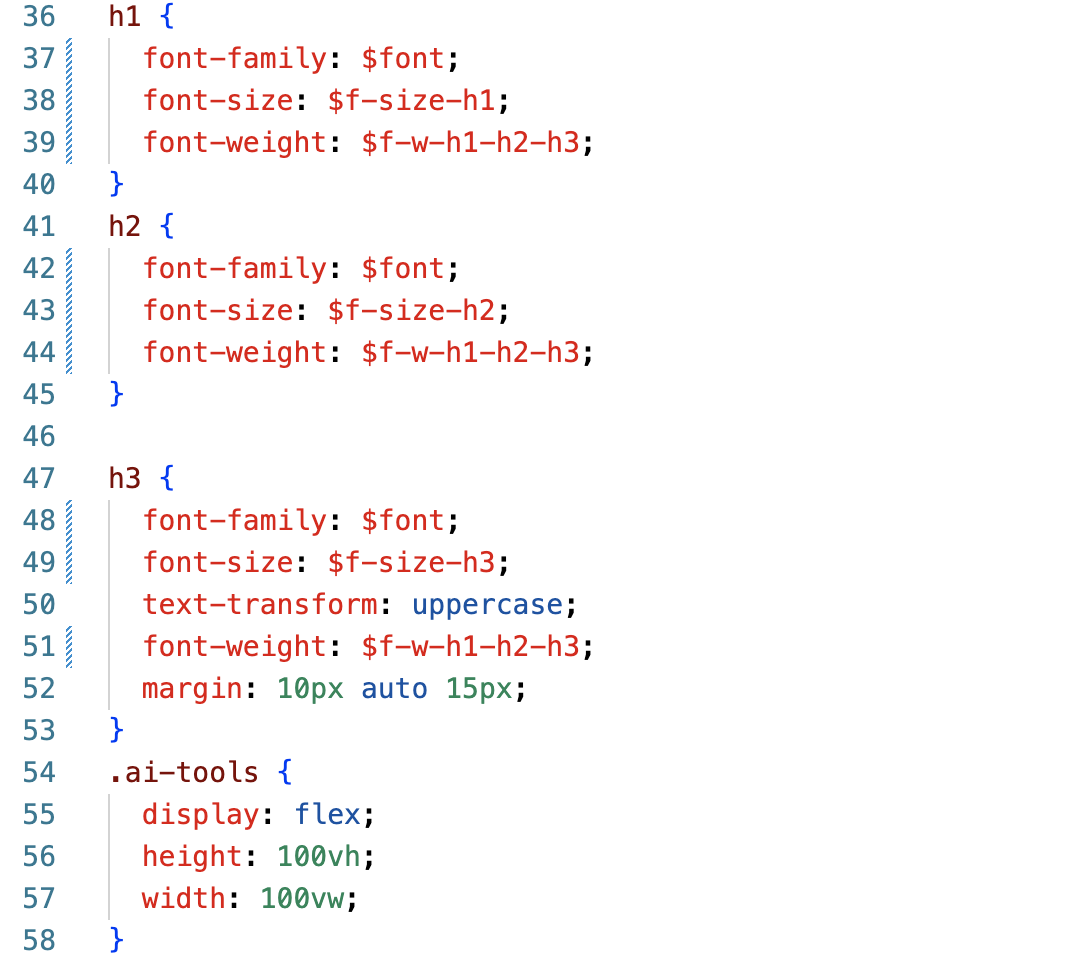
Структура HTML продумана так, чтобы облегчить интеграцию с серверной частью и минимизировать дублирование кода. Например, секция истории запросов может быть динамически обновлена через JavaScript, не требуя полной перезагрузки страницы. А использование функций url\_for гарантирует, что сервер всегда будет правильно подключать стили и скрипты, даже если структура проекта изменится.

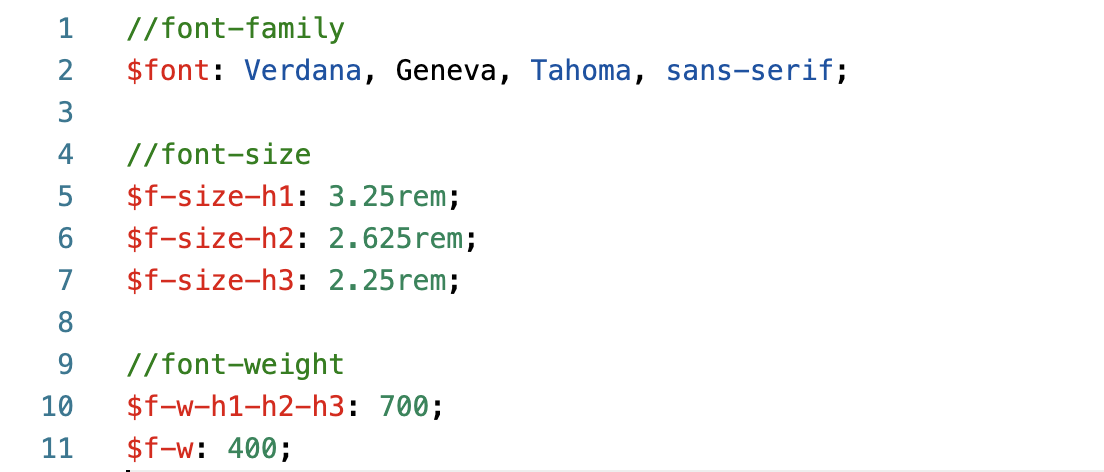
2.5.4 Управление стилями с помощью SCSS

В проекте используются «преимущества SCSS, такие как импорты, переменные и вложенность, что упрощает управление стилями и их масштабируемость» [3].

Файл index.scss начинается с базового сброса стилей, который обеспечивает единое отображение всех элементов на странице независимо от браузера.

SCSS позволяет разбить стили на отдельные модули, что делает код более структурированным и удобным для поддержки. В проекте используется директива @import, которая объединяет модули в единый файл:

****Также в файле index.scss есть использование стилей глобального класса, такой как «container» и стили по тегам h1, h2, h3. Их стили записаны с помощью переменных $font, $f-size-h1, $f-w-h1-h2-h3 и других, описанных в файле var.scss. Это улучшает читаемость и упрощает внесение изменений: достаточно изменить значение переменной в одном месте, чтобы это отразилось на всех заголовках.

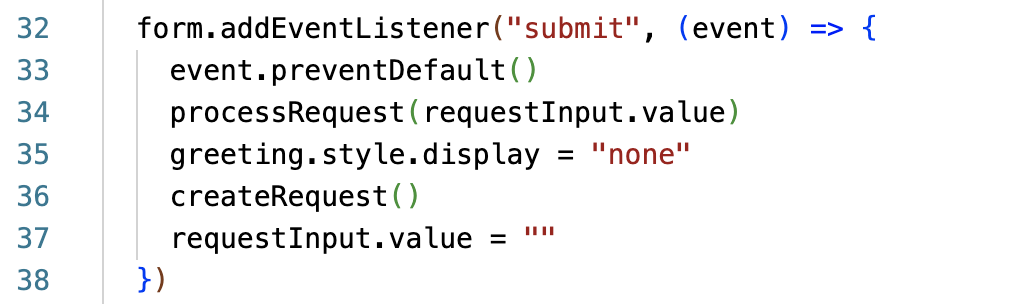
Файл \_var.scss:

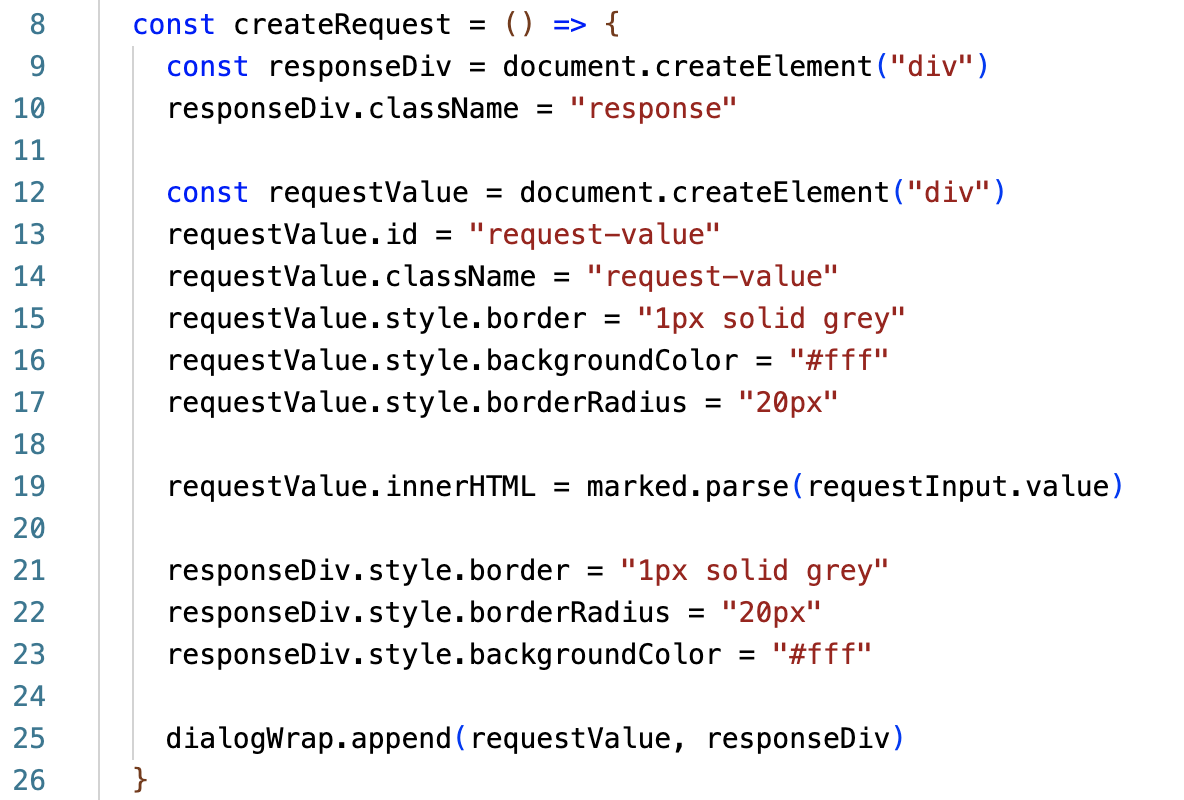
В остальных файлах я использовала вложенность, что тоже очень удобно.

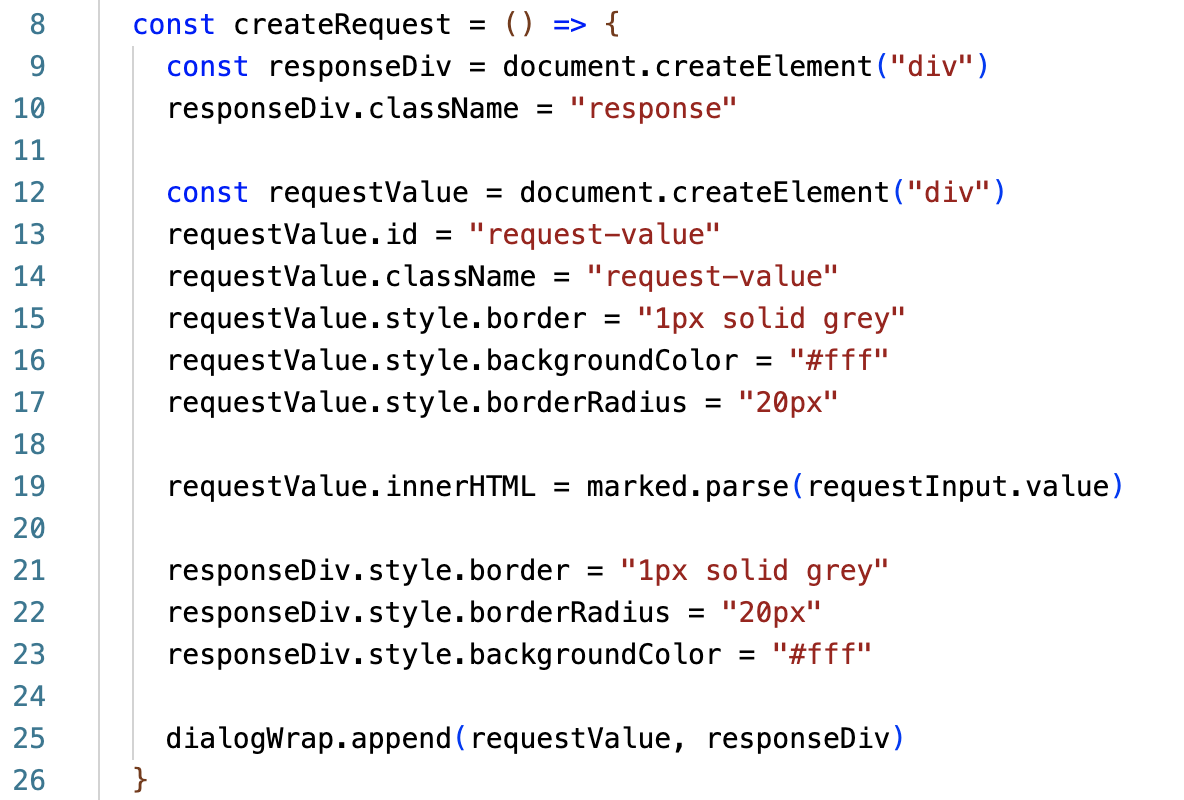
SCSS-файлы в проекте организованы таким образом, чтобы их можно было легко компилировать в один CSS-файл, подключаемый к HTML через Flask. Parcel автоматически отслеживает изменения в SCSS-файлах, компилируя их в готовый CSS: сначала все модули собираются в один файл index.scss, затем Parcel обрабатывает этот файл и генерирует скомпилированный index.css, и тогда готовый CSS-файл подключается к HTML.

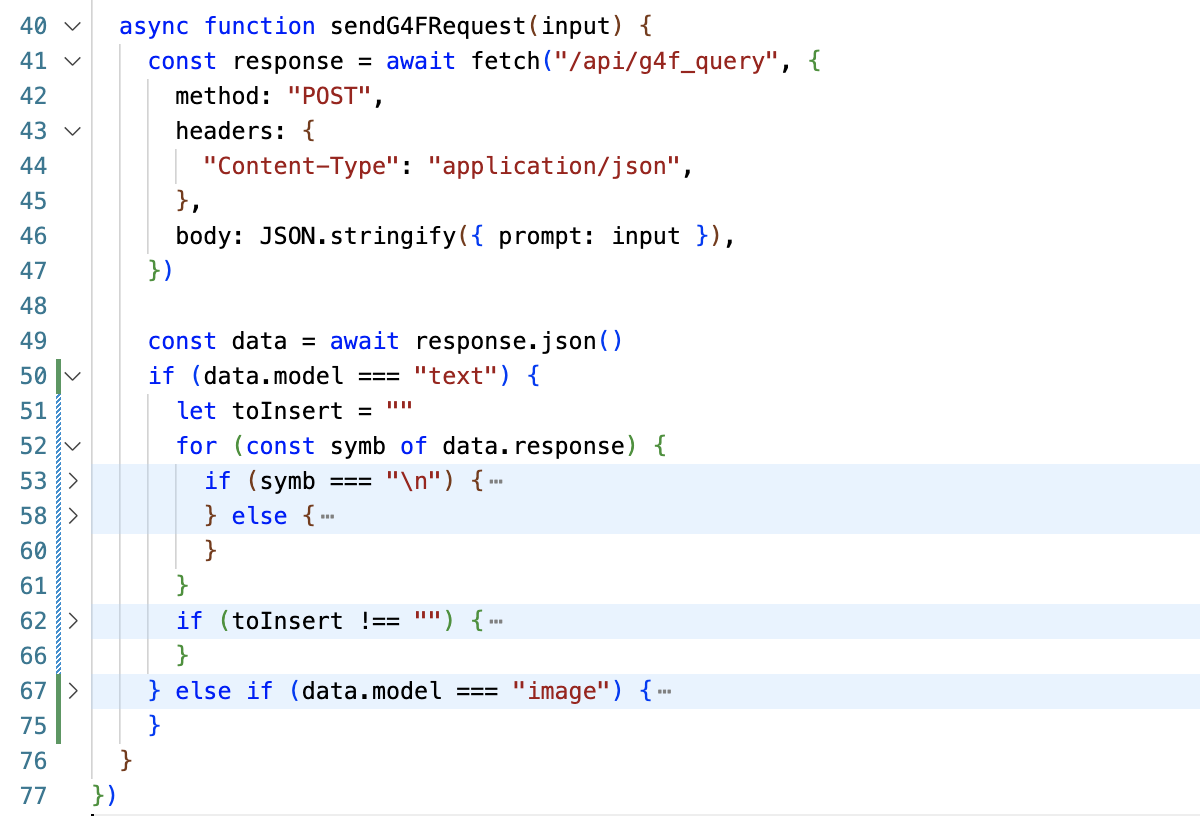
2.5.5 Реализация логики на JavaScript

JavaScript в проекте играет ключевую роль, обеспечивая интерактивность интерфейса, обработку пользовательских запросов и связь между клиентской и серверной частями через API.

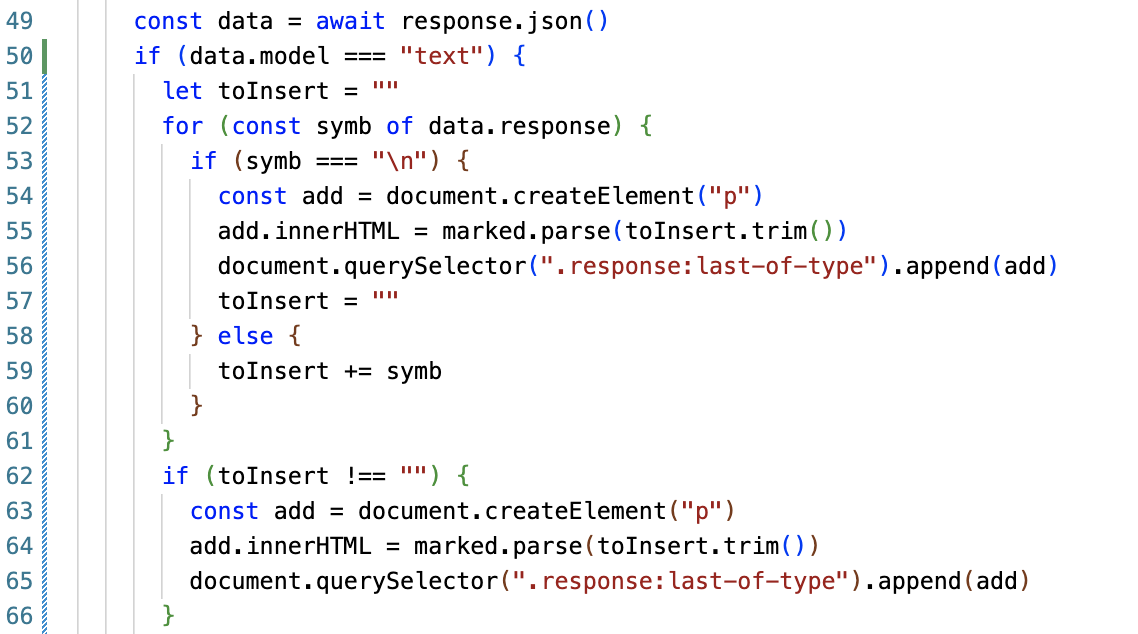
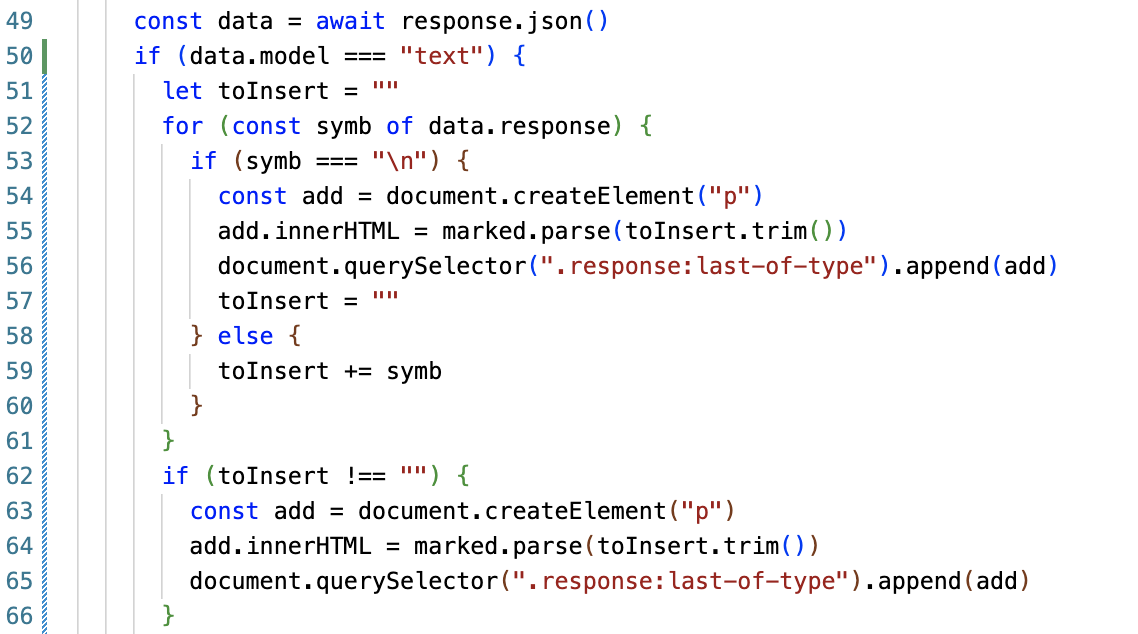
****Основной функционал приложения заключается в обработке запросов, которые пользователь вводит через текстовое поле.  
Для этого используется форма, отрабатывающая событие submit. Когда пользователь отправляет запрос, текст из поля ввода передается в функцию processRequest, прячется приветствие и на странице создается соответствующий визуальный элемент, представляющий этот запрос.

Функция createRequest создает блоки, которые визуально представляют запрос пользователя.

Каждый запрос оборачивается в блок с определенными стилями, обеспечивающими читабельность и визуальную согласованность интерфейса.

Взаимодействие между клиентской и серверной частями осуществляется через API. Когда пользователь вводит запрос, клиент отправляет его на сервер через POST-запрос. Через fetch осуществляется HTTP-запрос на сервер, передавая данные в теле запроса. **Затем** сервер возвращает JSON-объект с результатами обработки, и мы разбиваем ответ на части и добавляем на страницу в виде новых элементов.

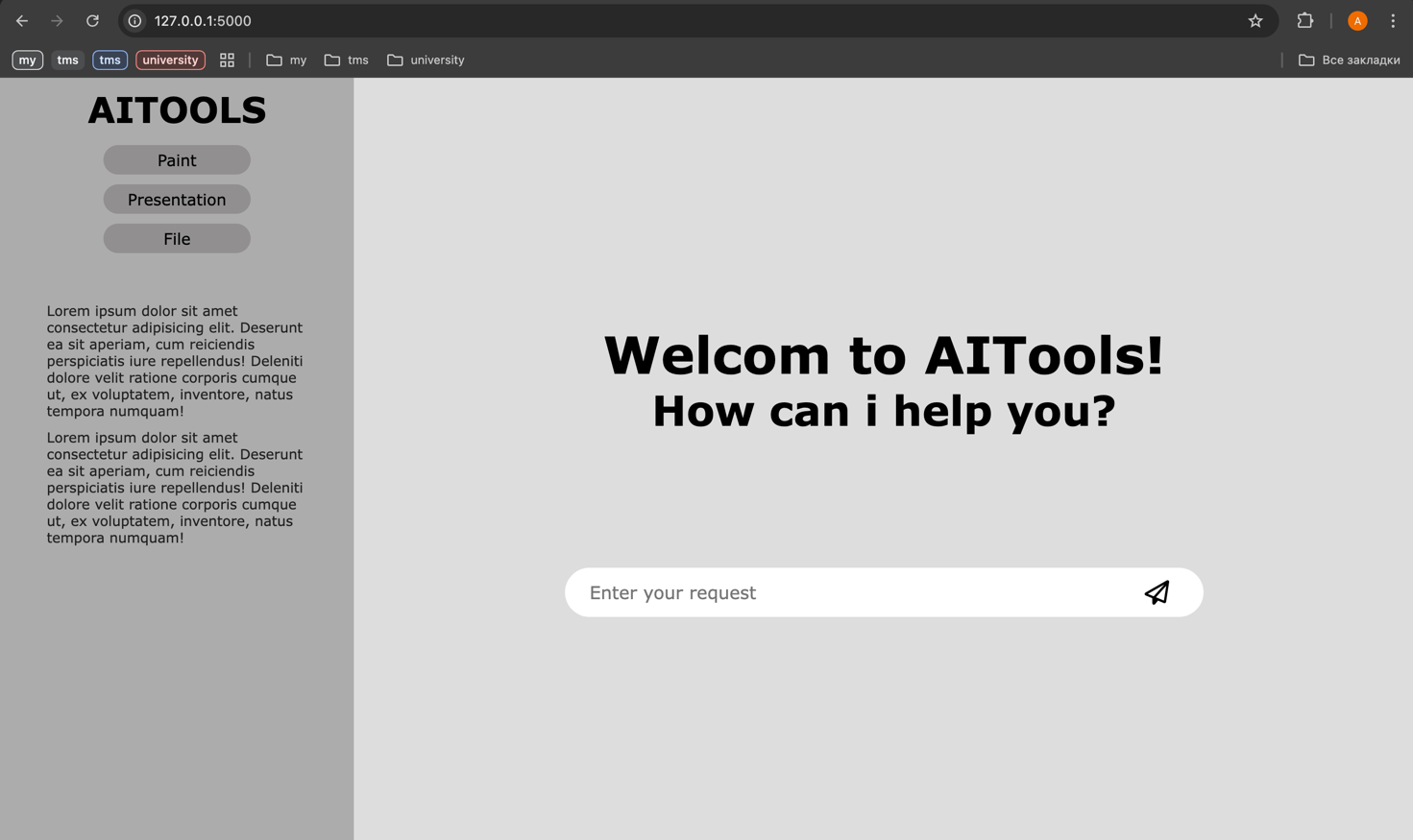
Для отображения текстового ответа создаются новые элементы, которые добавляются в последний созданный блок .response.

Этот код разбивает ответ на части, добавляя каждую строку в отдельный абзац. Это улучшает читаемость и удобство восприятия больших текстов.

А для отображения картинок создаются новые блоки, в которые добавляются изображения с ссылками, что нам приходят от сервера.

2.5.6 Результаты и демонстрация работы

Приложение представляет собой веб-сайт, где пользователь может взаимодействовать с нейросетями через графический интерфейс. Основной задачей было создание интуитивно понятного и функционального интерфейса, а также сайта, который объединяет данные из текстовых и графических запросов, отображает их в формате диалога, схожего с тем, как это реализовано на современных чат-платформах.

На рисунке 2.2 мы можем посмотреть на нашу главную страницу при входе.

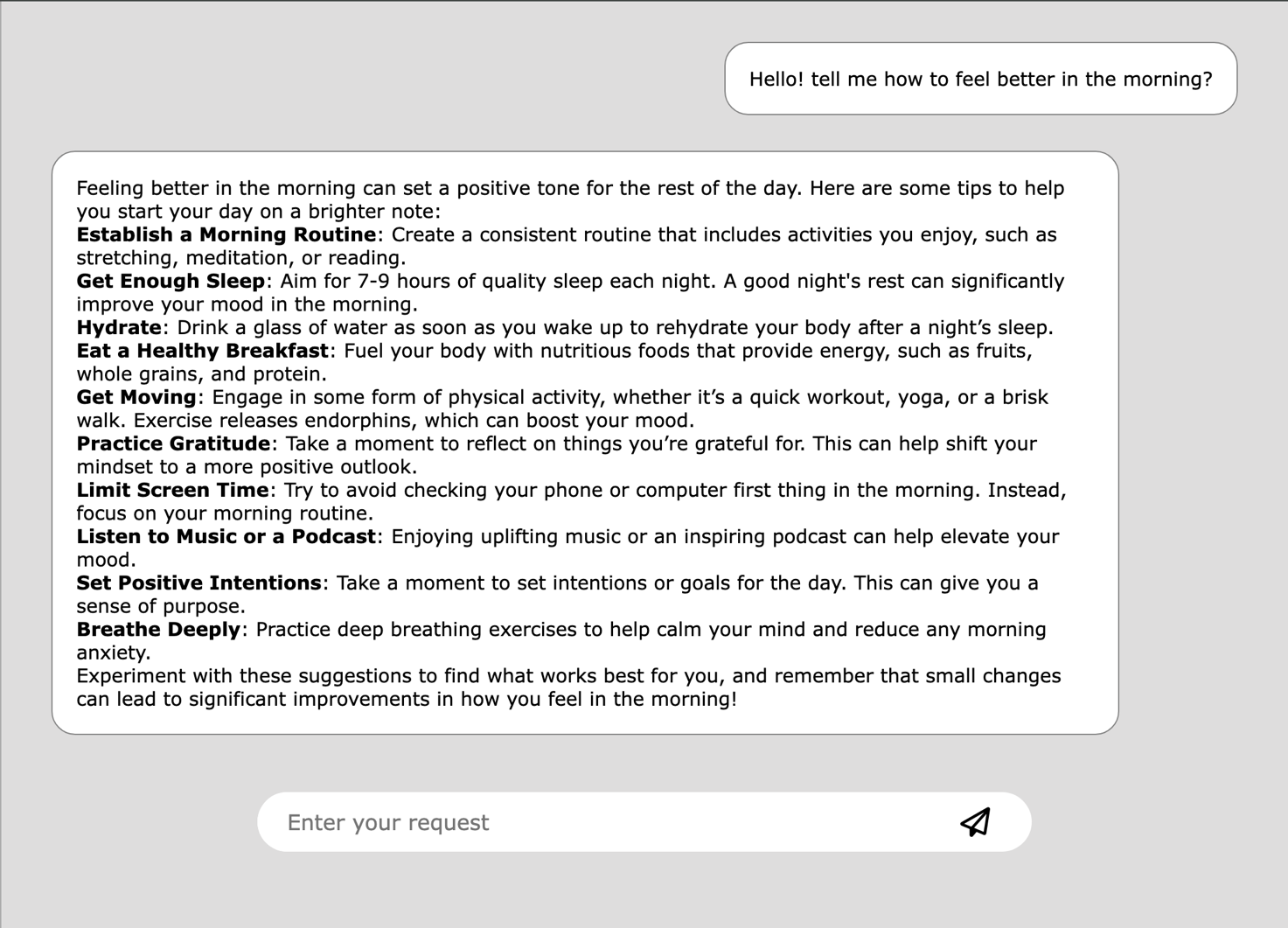
****Рисунок 2.2 – Главная страница веб-сайта

Рисунок 2.3 ­– Пример текстового запроса «Как чувствовать себя утром лучше?»

Как же выглядит наш диалог с текстовым запросом? Мы можем увидеть сверху на рисунке 2.3. Здесь мы запросили «Как чувствовать себя лучше утром?» и получили полный подробный ответ с использованием Markdown.

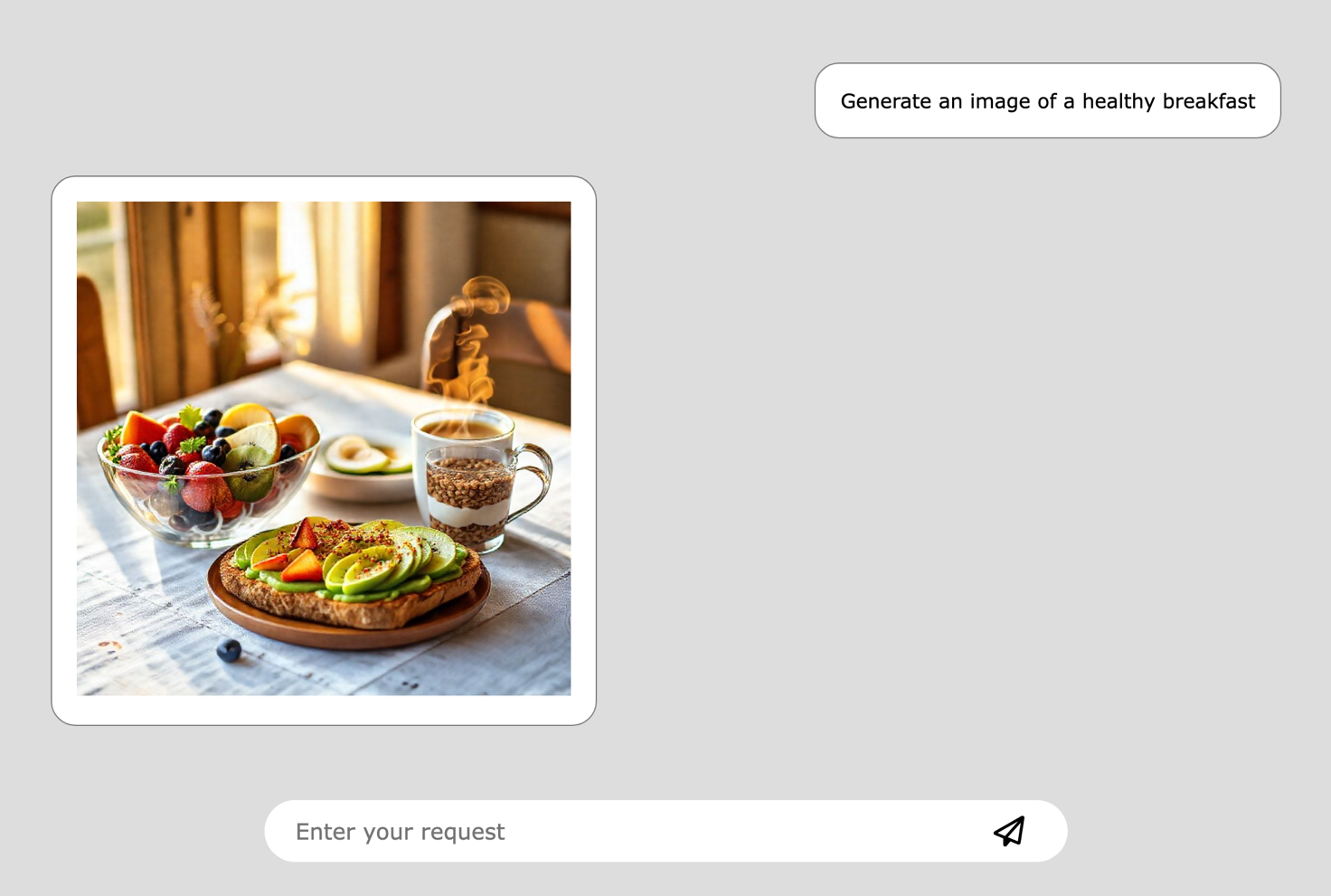
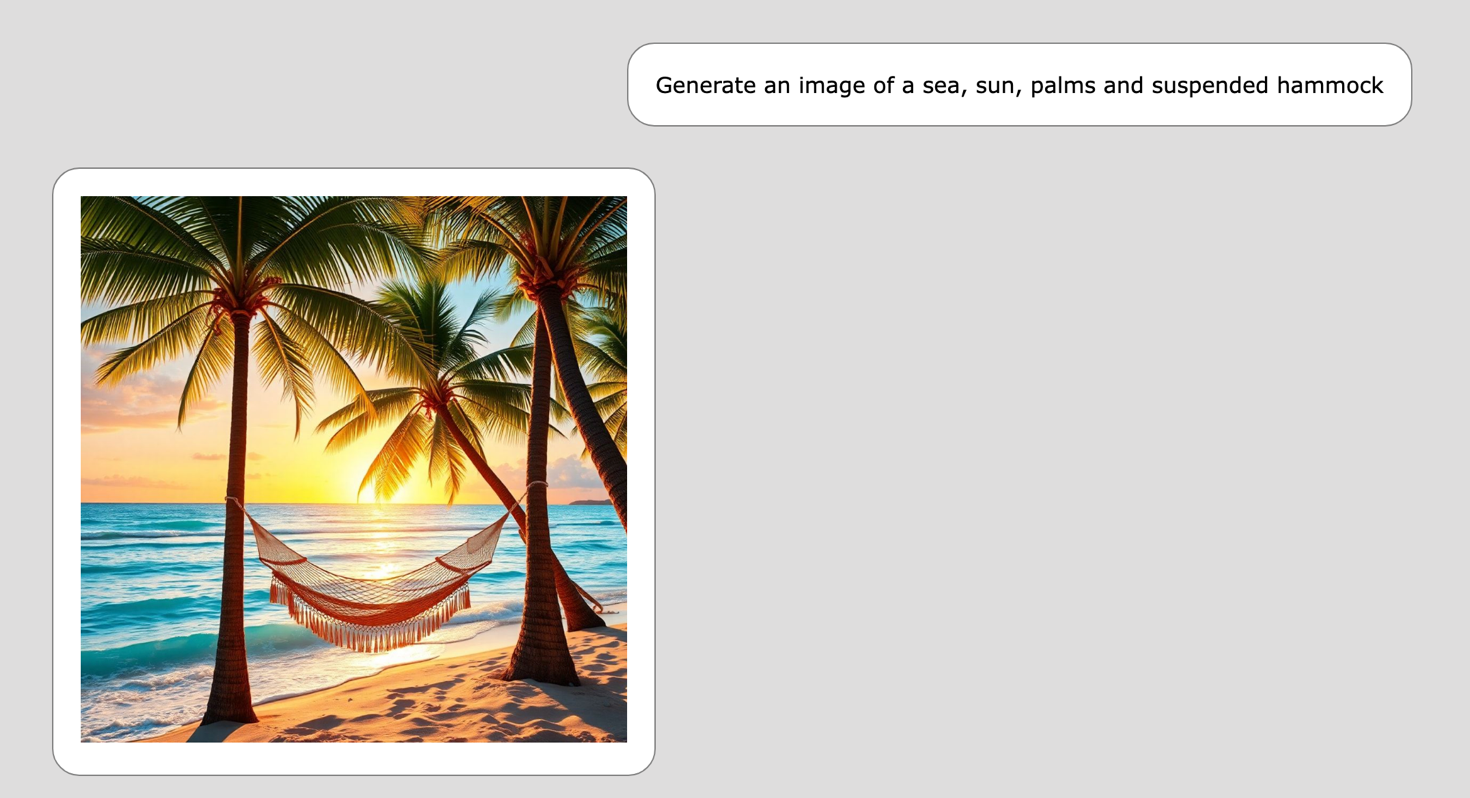
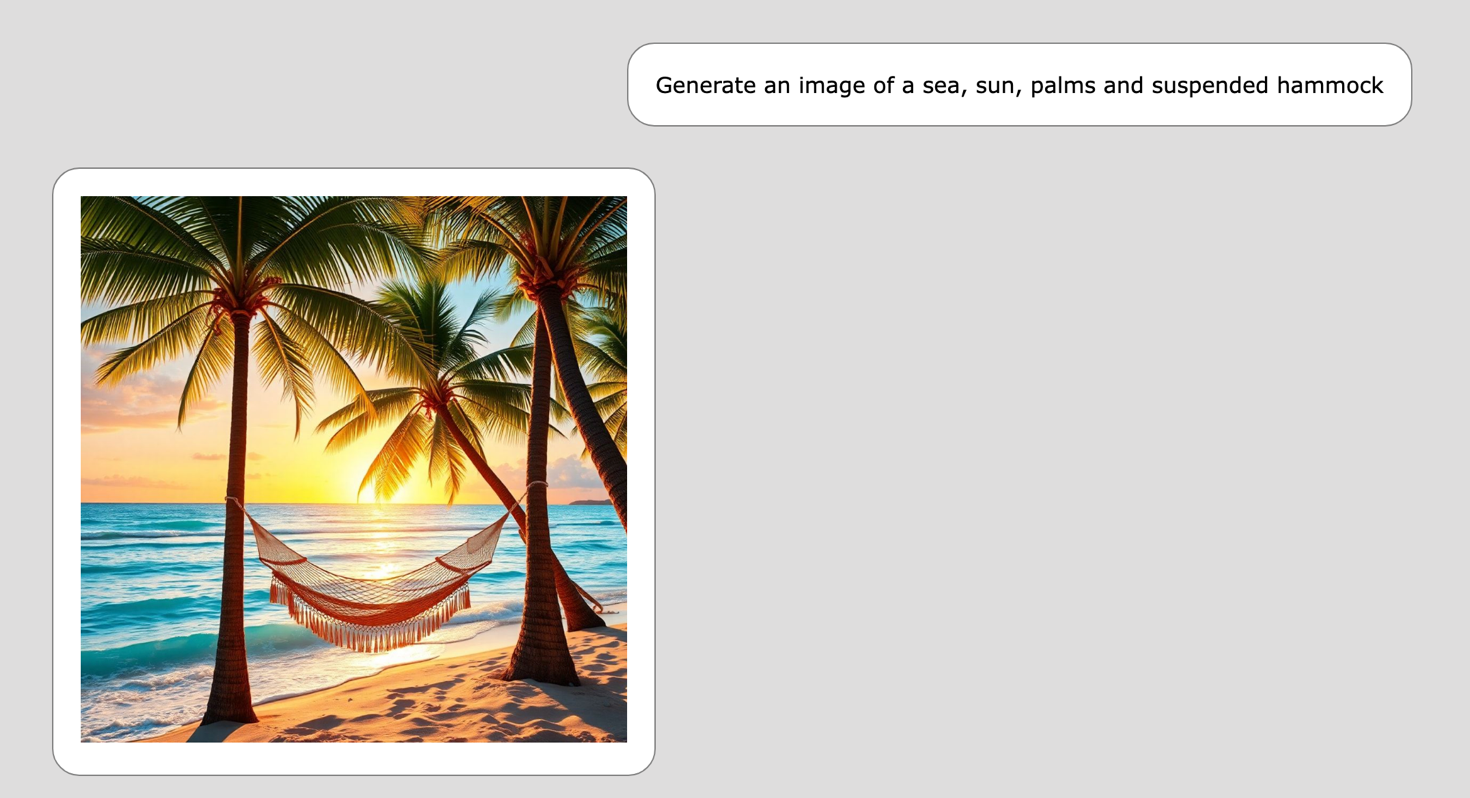
****Далее на рисунке 2.4 **мы видим запрос пользователя**, который он вводит в текстовом поле ввода, а именно: «Сгенерируй изображение полезного завтрака». Система обрабатывает запрос и генерирует соответствующее изображение. Затем полученный результат отображается ниже запроса в виде картинки. **В итоге справа мы видим** введённый пользователем запрос и **слева** генерируемое изображение.

Рисунок 2.4 ­– Генерация изображения «Полезный завтрак»

Ту же схему мы можем заметить на рисунке 2.5. **На нем изображен запрос пользователя**, который он вводит в поле ввода: «Сгенерируй изображение моря, солнца, пальм и подвешенного гамака». На сервере обрабатывается запрос и приходит соответствующая ссылка на изображение. Полученный результат отображается ниже в виде картинки.

 Рисунок 2.5 ­– Генерация изображения «Море, солнце, пальмы и подвешенный гамак»

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была разработана клиентская часть системы-агрегатора, способная обрабатывать запросы, получаемые от нескольких нейросетей, и отображать их результаты в диалоговом формате. Реализация проекта включала разработку и интеграцию интерфейса, обработку запросов с использованием JavaScript, а также оформление страницы с помощью SCSS и её сборку с помощью Parcel.

Созданный проект успешно объединяет данные из различных источников, обеспечивая удобный и интуитивно понятный интерфейс для пользователя. Все запланированные задачи, такие как сбор и отображение текстовых и графических данных, были выполнены. Пользователь может взаимодействовать с системой, отправлять запросы и получать результаты в реальном времени.

**Выполненная** работа соответствует современным стандартам веб-разработки. Использование SCSS и Parcel для сборки проекта позволило обеспечить модульность и поддерживаемость кода, а также адаптивность интерфейса. Реализация API-взаимодействия между клиентской и серверной частями демонстрирует высокий уровень интеграции современных технологий и подходов. Сравнение с лучшими достижениями в области разработки веб-приложений и систем на основе искусственного интеллекта показывает, что проект имеет потенциал для расширения и может стать основой для создания более сложных и функциональных решений.

В процессе реализации были выявлены некоторые недостатки, такие как необходимость улучшения обработки сложных запросов и обеспечения более гибкой работы с различными форматами данных, а также скорость получения ответов. Также требуются дополнительные оптимизации для повышения производительности при больших объемах данных.

Для дальнейшего улучшения проекта планируется интеграция дополнительных функций, таких как история запросов, авторизация и анализ данных, поддержка файлов и улучшение взаимодействия с сервером для увеличения скорости отклика. Можно также рассмотреть использование новых технологий и библиотек, что позволит улучшить функциональные характеристики и производительность системы. В перспективе проект может быть расширен с учетом новых требований и пользовательских потребностей.

Проект, в целом, соответствует современным требованиям веб-разработки и интеграции с нейросетями, что подтверждает его актуальность и перспективность для дальнейшего развития и использования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация Figma [Электронный ресурс]. ­– Режим доступа: <https://www.figma.com>. – Дата доступа: 15.11.2014.
2. Документация Parcel [Электронный ресурс]. – Режим доступа: <https://parceljs.org>. – Дата доступа: 24.11.2024.
3. SCSS – официальный сайт [Электронный ресурс]. – Режим доступа: [https://sass-lang.com](https://sass-lang.com" \t "_new). – Дата доступа: 24.11.2024.
4. Браун, И. Веб-разработка с применением Node и Express / И. Браун. – Санкт-Петербург : Питер, 2021. – 336 с.
5. Skillbox. Зачем сайту нужен дизайн [Электронный ресурс] // Жанна Травкина : образовательный портал. – Режим доступа: <https://skillbox.ru/media/design/>zachem\_saytu\_nuzhen\_dizayn/. – Дата доступа: 12.11.2024.
6. TeachMeSkills. Figma: что это такое и почему полезно знать [Электронный ресурс] // Миша Агеев : образовательный портал. – Режим доступа: https://teachmeskills.by/blog/figma-chto-eto-takoe-i-pochemu-polezno-znat. – Дата доступа: 25.11.2024.